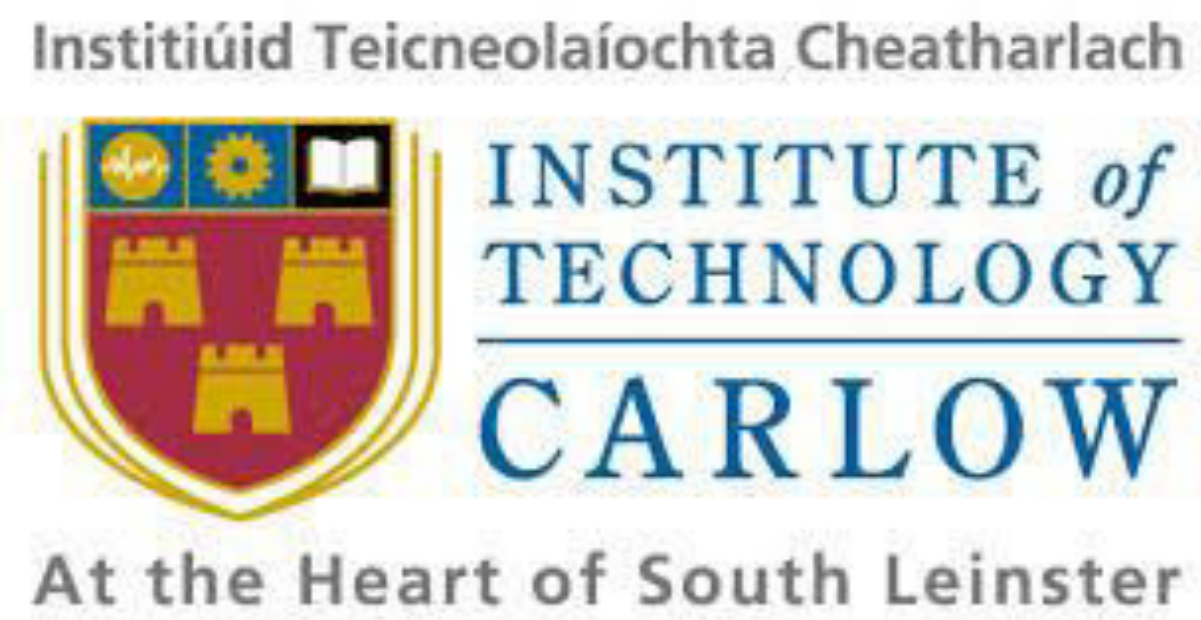
Project Report



Student: Ignas Usinskas
Student ID: C00166783
Supervisor: Greg Doyle
Date: 15/04/2016

Table of Contents

Introduction

The purpose of this document is to discuss the final product that was built during this project. Evaluate what the student was able to achieve and what he was not able to achieve. This document will show what the student has learned from this project and what he would do differently next time.

Project Description

This project – Sentiment analysis of social media (Twitter) is about analysing tweets using sentiment analysis and determining if they are positive, negative or neutral. The product at the end of this project is a web application hosted on a private server.

The not web part of this web application streams tweets, analyses them using NLTK and sklear libraries for python and stores them in MySQL database which is hosted on the same server. The web part of this application provides a search page for users where they can enter a search query and get results for that search query.

Problems encountered

- Scoping – The original idea was to stream 50 tweets according to the search term the user entered, evaluate tweets and display them in the results page. The problem was that I could not pass the search term to the stream filter due to scoping issues. Problem was fixed with tweets being streamed every day based on location rather than the search term. Tweets are evaluated and stores in the database. When user enters a search term, tweets are pulled from the database and the ones containing the search term are displayed in the results page. Even if this solution is not as practical as the original idea, it produces results that can be used.
- Sentiment analysis – At first I was barely able to do sentiment analysis using only one classifier which proved inaccurate and unreliable results. Thankfully to good tutorials found online, I was able to combine seven different classifiers which increased accuracy and reliability.
- Graphing the results – Live graphing using matplotlib library for python proved to be too complex as I could not display the graph in the results page. The live graphing would stop the entire application and open up a new window to display the graph. To fix this I got rid of the live graphing and decided to save the graph as a picture, when the results are displayed – the picture of graphed results is displayed at the top of the page.
- Updating graph picture – at first, the picture of the graphed results would be made every time when application would be started. This was due to the how imported modules in python work. If code in the module is not in a function, it executes as soon as the application starts and can never be executed again. I fixed it by modifying the code and putting it into a function.
- Inaccurate graphs – if the data passed to the graph was less than 1000 evaluations, the graph would not be made and if the data was sufficient, graph would show inaccurate results. This was due to overcomplicating the code by storing data to be graphed in a text file, then pulling that data in another module. Problem was fixed by discarding separate module for creating the graph, putting code in search.py file and passing variables to be graphed to the function where graph is plotted and saved.

- Run script for a certain amount of time, each day – Proved to be too time consuming, and had to be dropped it in the end.
- Run the not flask side of the application online on the server – this problem could not be solved, because of the little time I had left and had to be improvised by the streaming being done locally and the data from the local database being exported to the database on the hosting server. This problem arises, when streaming is done, all exceptions except for the keyboard interrupt are passed and sentiment analysis using a lot of CPU power making the code run slow and burn though the CPU allowance very quickly.

What I have achieved

The requirements for this project were simple: stream tweets and evaluate them using sentiment analysis techniques. What I have achieved:

- Streaming tweets using tweepy library for python.
- Sentiment analysis of tweets using NLTK supervised machine learning.
- Deleting tweets that are over 24 hours old from the database.
- Storing analysed tweets in MySQL database.
- Search page provided for users.
- Searching analysed tweets.
- Minor error checking and appropriate error messages.
- Displaying results:
    - Graph.
    - Search term.
    - Total evaluation.
    - Tweet count.
    - Tweets used in the evaluation.

What I did not achieve

- Live Graphing of the results.
- Streaming tweets based on selected location, now tweets are streamed only from Ireland.
- Streaming tweets based on search term.
- Saving tweets over long periods of time so the change in people opinion can be seen over time. This could not be done due to the terms and conditions of Twitter API. Tweets are not allowed to be stored for more than 24 hours.
- Run script for a certain amount of time, each day.
- Streaming, analysing and storing tweets on the server. Streaming on pythinanywhere.com server seems to be impossible as it is very CPU dependant and pythonanywhere.com limits CPU usage by providing only 100 seconds of CPU allowance for users with free accounts. Thus the streaming has to be done locally and the data from the local database has to be exported to the database hosted on pythonanywhere.com

What would I do differently if starting again

- Instead of using bootstrap to make a simple GUI, I would try and make a more appealing and feature rich GUI.

- Would take more time to graph the results and try to make the graph as appealing as possible.
- Instead of opening new pages for each task, would do a complete website just for this application.
- Try to merge the application into one so as to not have two processes running separately.
- Handle the exceptions better as right now all exceptions except key interrupt are passed so to prevent crashing when bad characters are encountered in tweets. This puts a lot of strain on the application making it very CPU dependant.
- Take more time to put it on the server.
- Choose different server than pythonanywhere.com as this server is very bad for this application, because of the CPU usage limitations.

Differences

The way the application would operate has changed entirely. At first the idea was to download tweets based on the search term entered by the user. As it proved to be too time consuming to deal with scoping issues, the application had to change. Now tweets are downloaded once every day, evaluated and stored in the database while tweets that were stored in the database for over 24 hours are removed. When users search for tweets, tweets are pulled from the database. If a tweet contains the search term – it is displayed in the results page.

Different database, MariaDB was the initial choice, but was reconsidered latter and replaced with MySQL because of popularity and supportability on private servers.

The original idea was to have a single process to do all the work – stream tweets, analyse them, store then in the database, display search page with the results. As it proved very challenging to achieve this – the application had to be split into two. One part is the non-user part where the application streams, analyses and stores tweets into the database without the user knowing about it. The other part is the web application which allows the user to search through stored tweets and if the tweets contain the search term – they are displayed in the results page.

Module descriptions

This application consists of two separate parts that work on different tasks to achieve the desired functionality. The application is made up of seven different modules.
First part, which is the streaming side of the application, consists of six modules:

1. Sent.py – is the main body of this part of the application, which connects the other modules and runs to stream tweets from twitter.
2. Sentiment.py – this module creates and trains classifiers which are used in sentiment analysis. The classifiers are pickled out once they are trained so they would not need to be retrained every time the sent.py is run.
3. Sentiment_module.py – is almost identical to the module described above, differing only in the pickled classifiers being used instead of training classifiers every time this module is executed. This is the sentiment analysis part of the application which is called by sent.py when tweets are streamed.
4. Positive.py – is a module which contains a list of 500 positive tweets and their sentiment evaluations. This module is used in training the classifiers.

5. Negative.py – is a module which contains a list of 500 negative tweets and their sentiment evaluations. This module is used in training the classifiers.
6. Neutral.py – is a module which contains a list of 500 neutral tweets and their sentiment evaluations. This module is used in training the classifiers.

Second part of the application is the flask side of it. This part provides GUI for users to use. It consist of one module and four templates:

1. Search.py – is the main body of this part of the application which is hosted online on the server. This module connects to MySQL database when user searches for something and displays relevant tweets in results page.
2. Search.html – is the template used by search.py to display the search page of the application.
3. Results.html – is the template used by search.py to display the results page with all the information in it.
4. Empty_string.html – is the template used by search.py to display an error page when no tweets were found to contain the searched keyword.
5. Results.html – is the template used by search.py to display an error page when user forgot to input anything in the search field.

Testing

Streaming was tested by leaving it on for long periods of time. Few hours of streaming seems to be the limit before Twitter disconnects it.

Streaming was attempted on the server, but as it is heavy on the CPU – it is not possible to stream on the current server that the application is hosted on.

Web application side was thoroughly tested both, locally and on the web server to make sure everything works.